Primary Key and **Foreign Key** down *cleanly*, *practically*, and **MySQL-style**, with real examples you'd actually use

---

1 Primary Key (PK)

What it is

A **Primary Key**:

- Uniquely identifies **each row** in a table
- **Cannot be NULL**
- **Cannot repeat**
- There is **only one primary key per table**
- Often used with `AUTO_INCREMENT`

Why it matters

Without a primary key:

- You can't reliably identify a specific row
- Relationships with other tables break
- Indexing and performance suffer

Example: `students` table

```
CREATE TABLE students (
    student_id INT AUTO_INCREMENT,
    name VARCHAR(100),
    email VARCHAR(100),
    PRIMARY KEY (student_id)
);
```

Explanation

Column	Meaning
`student_id`	Primary key — unique ID for each student
`AUTO_INCREMENT`	MySQL automatically generates a new number
`PRIMARY KEY`	Ensures uniqueness + not null

Sample data

```
student_id | name      | email
-----------|-----------|-----------------
1          | Rahul     | rahul@mail.com
2          | Ananya    | ananya@mail.com
```

✔ Even if two students have the same name, `student_id` is always unique.

---

2 Foreign Key (FK)

What it is

A **Foreign Key**:

- Creates a **relationship between two tables**
- Refers to a **Primary Key in another table**
- Maintains **referential integrity**

Why it matters

It prevents:

- Orphan records (data that shouldn't exist)
- Invalid references
- Data inconsistency

Example: `courses` table

```
CREATE TABLE courses (
    course_id INT AUTO_INCREMENT,
    course_name VARCHAR(100),
    PRIMARY KEY (course_id)
);
```

Example: `enrollments` table (Foreign Key usage)

```
CREATE TABLE enrollments (
    enrollment_id INT AUTO_INCREMENT,
    student_id INT,
    course_id INT,
    PRIMARY KEY (enrollment_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Explanation

Column	Role
`enrollment_id`	Primary key of enrollments
`student_id`	Foreign key → `students.student_id`
`course_id`	Foreign key → `courses.course_id`

✔ A student **must exist** before enrollment
✔ A course **must exist** before enrollment

---

3 How Primary & Foreign Keys Work Together

Relationship

```
students (1)  ----<  enrollments  >----  (1) courses
```

- One student → many enrollments
- One course → many enrollments

Valid Insert (works)

```
INSERT INTO enrollments (student_id, course_id)
VALUES (1, 2);
```

✔ Student `1` exists
✔ Course `2` exists

Invalid Insert (fails □)

```
INSERT INTO enrollments (student_id, course_id)
VALUES (99, 2);
```

□ Error: student `99` does not exist

the same table, but now **with rules** (`NOT NULL`, `ON DELETE`, `ON UPDATE`) and explain **why each rule is used** in MySQL.

`enrollments` Table with Rules

```
CREATE TABLE enrollments (
    enrollment_id INT AUTO_INCREMENT,
    student_id INT NOT NULL,
    course_id INT NOT NULL,
    enrollment_date DATE DEFAULT CURRENT_DATE,

    PRIMARY KEY (enrollment_id),

    CONSTRAINT fk_student
        FOREIGN KEY (student_id)
        REFERENCES students(student_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT fk_course
        FOREIGN KEY (course_id)
        REFERENCES courses(course_id)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);
```

Explanation of Each Rule

1□ `PRIMARY KEY (enrollment_id)`

Rules applied automatically

- Must be **unique**
- Cannot be **NULL**
- Indexed automatically

- One primary key per table

Why?
Each enrollment must be uniquely identifiable.

2️⃣ `student_id INT NOT NULL`

Rule

- `NOT NULL` → enrollment **must have a student**

Why?
An enrollment without a student makes no sense.

3️⃣ `course_id INT NOT NULL`

Rule

- `NOT NULL` → enrollment **must have a course**

Why?
Prevents meaningless records.

4️⃣ `FOREIGN KEY (student_id)`

```
REFERENCES students(student_id)
ON DELETE CASCADE
ON UPDATE CASCADE
```

Rules Explained

Rule	Meaning
`REFERENCES`	Must match an existing student

Rule	Meaning
`ON DELETE CASCADE`	If student is deleted → delete enrollments
`ON UPDATE CASCADE`	If student_id changes → auto-update here

 Keeps data clean
 Prevents orphan enrollments

---

5 `FOREIGN KEY (course_id)`

```
REFERENCES courses(course_id)
ON DELETE RESTRICT
ON UPDATE CASCADE
```

Rules Explained

Rule	Meaning
`ON DELETE RESTRICT`	Cannot delete course if enrollments exist
`ON UPDATE CASCADE`	Updates course_id everywhere

Why RESTRICT here?
You usually **don't want to delete a course** if students are enrolled.

---

6 Named Constraints (Best Practice)

```
CONSTRAINT fk_student
CONSTRAINT fk_course
```

Why name them?

- Easier debugging

- Clear error messages
- Easier to drop or modify later

Example: What the Rules Do

✔ Valid Insert

```
INSERT INTO enrollments (student_id, course_id)
VALUES (1, 2);
```

□ Invalid Insert (student doesn't exist)

```
INSERT INTO enrollments (student_id, course_id)
VALUES (99, 2);
```

Error: foreign key constraint fails

□ Delete Course (restricted)

```
DELETE FROM courses WHERE course_id = 2;
```

Error: course is still in use

✔ Delete Student (cascades)

```
DELETE FROM students WHERE student_id = 1;
```

✔ Student deleted
✔ All enrollments deleted automatically

Summary of Rules Used

Rule	Purpose

Rule	Purpose
`PRIMARY KEY`	Unique row identification
`NOT NULL`	Mandatory data
`FOREIGN KEY`	Relationship enforcement
`CASCADE`	Auto-sync changes
`RESTRICT`	Prevent unsafe deletes
`DEFAULT`	Auto values